WEST

Generate Collection

L7: Entry 2 of 6 File: USPT Jul 27, 1999

US-PAT-NO: 5930509
DOCUMENT-IDENTIFIER: US 5930509 A

TITLE: Method and apparatus for performing binary translation

DATE-ISSUED: July 27, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Yates; John S.	Needham	MA	N/A	N/A
Tye; Steven Tony	Hopkinton	MA	N/A	N/A
Hookway; Raymond J.	West Boylston	MA	N/A	N/A

US-CL-CURRENT: 717/7; 717/5

CLAIMS:

What is claimed is:
1. A method executed in a computer system for performing binary translation of a first binary image associated with a first computer architecture having a first instruction set to a second binary image associated with a second computer architecture having a second instruction set, the method comprising the steps of:
translating, in accordance with profile execution information produced from a runtime execution of said first binary image by a runtime interpreter, a first portion of said first binary image to a second portion of said second binary image by producing a plurality of intermediate representations, each of said plurality of intermediate representations including one or more intermediate elements, each of said intermediate elements associated with a set of invariant characteristics remaining constant throughout said translating step, said set of invariant characteristics including at least two invariant characteristics, said at least two invariant characteristics including correspondence to an instruction included in said second instruction set, and correspondence to an instruction in said first instruction set, said translating step including the substeps of:
producing an initial intermediate representation of said first portion;
producing a final intermediate representation of said first portion, said initial and said final intermediate representations being of said plurality of intermediate

representations; and
optimizing said initial intermediate representation, substeps of optimizing and translating being intermixed to produce said final intermediate representation; and
generating said second portion of said second binary image using said final intermediate representation, said profile execution information characterizing said runtime execution of said first binary image;
wherein said initial and final intermediate representations include a list of instruction code cells, each of said instruction code cells including an instruction opcode and one or more code cell operands, said first computer architecture being a complex instruction set computer and said second computer architecture being a reduced instruction set computer, and wherein a machine instruction included in said first portion of said first binary image corresponds to one or more instruction code cells in said initial intermediate representation, and wherein said step of producing said initial intermediate representation includes the substeps of:
associating an exception bit flag with each of said instruction code cells;
initializing, as indicated in an opcode exception table, each of said exception bit flags, each of said exception bit flags being set to a bit value of 1 when said opcode exception table indicates a runtime exception can be generated when a machine instruction, which is in said first instruction set and corresponds to the instruction opcode, is executed, otherwise each of said exception bit flags being set to a bit value of 0, said opcode exception table being indexed by instruction opcode and having an entry for each instruction opcode corresponding to an instruction in said first instruction set;
examining each machine instruction included in said first portion of said first binary image to determine memory operands, each of said memory operands representing a memory location;
determining for each of said memory operands an effective address formation to reference a corresponding memory location;

selecting for each of said memory operands one of a load operation or a store operation, the load operation being used to read from the corresponding memory location, and the store operation being used to write to the corresponding memory location;
determining for each machine instruction a functional operation performed by said each machine instruction;
generating, for each of said memory operands, a first instruction code cell and a second instruction code cell, said first instruction code cell computing the corresponding to the effective address formation, said second instruction code cell performing said selected one of said load operation or said store operation;
generating, for said each machine instruction, a third instruction code cell corresponding to said functional operation performed by said machine instruction; and
recording with each of said instruction code cells a first binary image address corresponding to the machine instruction from which said each instruction code cell is generated.

2. The method of claim 1, wherein said translating step further includes the step of optimizing said initial intermediate representation producing another intermediate representation used to produce said final intermediate representation.
3. The method of claim 2, wherein said initial, final and other intermediate representations each comprise a list of instruction code cells, each of said instruction code cells including an instruction opcode and one or more associated operands, said instruction opcode being one of a set of instruction opcodes comprising opcodes corresponding to source instructions associated with said first computer architecture and destination instructions associated with said second computer architecture.
4. The method of claim 1, wherein said initial intermediate representation comprises one or more basic blocks of instruction code cells, each of said basic blocks comprising one or more instruction code cells having no entry or exit between code cells of said each basic block, and wherein a condition code bit mask is associated with each of said instruction code cells of said initial intermediate representation, and wherein said substeps of optimizing and translating include condition code processing comprising the steps of:
performing, for each of said basic blocks, data flow analysis using said condition code bit masks associated with said instruction code cells in said each basic block to produce local summary condition code information associated with said each basic block;
determining, for each of said basic blocks using said summary condition code information associated with said each basic block, global condition code information identifying a portion of said condition codes which are referenced in other basic blocks; and
updating the initial intermediate representation to include a representation of said condition codes and references to said condition codes within and between basic blocks as determined, respectively, by said local summary condition code information and said global condition code information.
5. The method of claim 4, wherein said step of updating said initial intermediate representation includes the steps of:
adding to the initial intermediate representation one or more instruction code cells which sets said each condition code; and

adding to the initial intermediate representation one or more other instruction code cells which uses said each condition code.
6. The method of claim 1, wherein said step of translating includes register processing, said register processing comprising the steps of:
determining which operands of instruction code cells are partial register operands;
replacing each of said partial register operands with an entire register operand;
adding one or more instruction code cells to said first intermediate representation to produce a result using said entire register operand that is equivalent to using said partial register operand; and

determining and updating any data dependencies of said partial register operands.

7. The method of claim 1, wherein said step of translating includes performing floating-point peephole optimization processing which replaces a first group comprising four instruction code cells with a single other instruction code cell, said first group of instruction code cells corresponding to machine instruction in said first instruction set, said single other instruction code cell corresponding to a machine instruction in said second instruction set, and wherein said first group of instruction code cells comprises code cells corresponding to four machine instructions as follows:

a floating point test instruction that tests a floating point value and accordingly sets one or more status values;

a store instruction which stores said status values to a register;

a test instruction which compares the register containing the status values to a bit mask to determine a condition, and sets condition codes accordingly; and

a conditional branch instruction transferring control to one instruction upon the condition being satisfied, otherwise control is transferred to another instruction.

8. The method of claim 1, wherein said translating step includes performing floating point register stack addressing processing that replaces a first group of one or more instruction code cells with a second group of one or more other instruction code cells, said first group of instruction code cells corresponding to machine instructions in said first instruction set which push and pop floating point values used as operands from the stack register.

9. The method of claim 1, wherein said initial intermediate representation comprises one or more basic blocks of instruction code cells, each of said basic blocks comprising one or more instruction code cells having no entry or exit between code cells of said each basic block, and wherein said step of translating includes performing local and global optimizations, said local optimizations being performed within a basic block and said global optimizations being performed between two or more basic blocks.

10. The method of claim 1, wherein said step of translating includes performing exception handler processing that includes the steps of:

examining each instruction code cell in said initial intermediate representation to determine which operands cause exceptions to occur;

determining and recording a mapping of registers for each instruction code cell from the second instruction set to first instruction set, said mapping being used during execution of said second binary image if an exception occurs and being included in said second binary image; and

recording in said second binary image an intervening runtime user exception handler to which execution control is passed when a runtime exception occurs, said intervening runtime user exception handler mapping a first context to a second context which is passed to another exception handler, said first context including registers in said second instruction set, said second context including registers in said first

instruction set, said intervening runtime user exception handler using said mapping to map said first context to said second context.

11. The method of claim 1, wherein said step of translating includes final processing comprising the steps of:

replacing instruction opcodes corresponding to operations in said first instruction set with other instruction opcodes corresponding to operations in said second instruction set;

replacing 32-bit operands with 64-bit sign-extended operands;

performing interimage call processing for each translation unit that is called from another translation unit, said each translation unit being not defined within said first binary image, said interimage call processing causing runtime execution control to be passed to said runtime interpreter when performing said interimage call at a subsequent runtime; and

performing intra-image call processing for calls between translation units in said first image file, intra-image call processing including, for each intra-image call made from a first translation unit to a second translation unit, the step of:

replacing one or more instruction code cells corresponding to said intra-image call with other instruction code cells causing runtime execution control to directly transfer from said first translation unit to said second translation unit without passing runtime execution control to said runtime interpreter when performing said intra-image call.

12. The method of claim 11, wherein said step of replacing 32-bit operands with 64-bit sign-extended operands uses local data flow analysis information to locate operand references and definitions in said initial intermediate representation.

13. The method of claim 11, wherein said intra-image call processing is performed for a first group of one or more instruction code cells in said initial intermediate representation corresponding to one or more instructions in said first instruction set for implementing one of a program counter-relative call or absolute call from a first translation unit to a second translation unit, and wherein said first group of instruction code cells is replaced with a second group of instruction code cells corresponding to one or more instructions for a program counter-relative call in said second instruction set.